Introduction to OpenSCAD

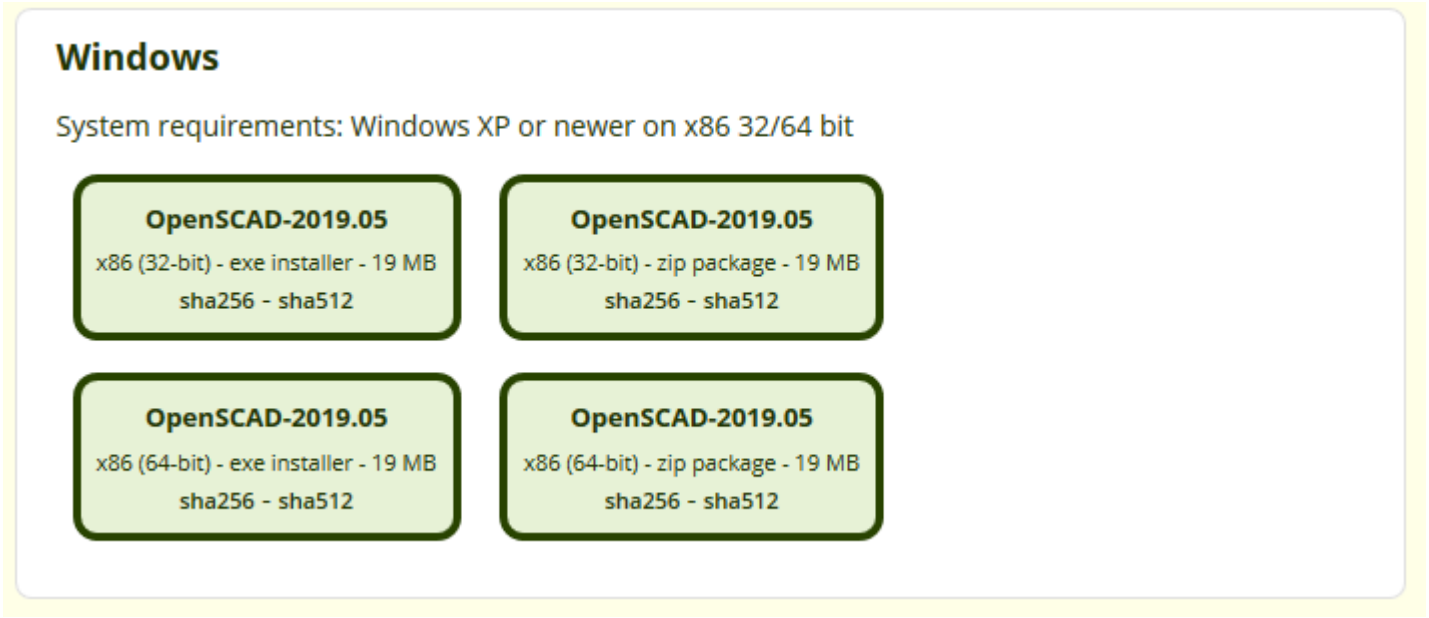
Adam Watts

June 2020

What is OpenSCAD and why should I care?

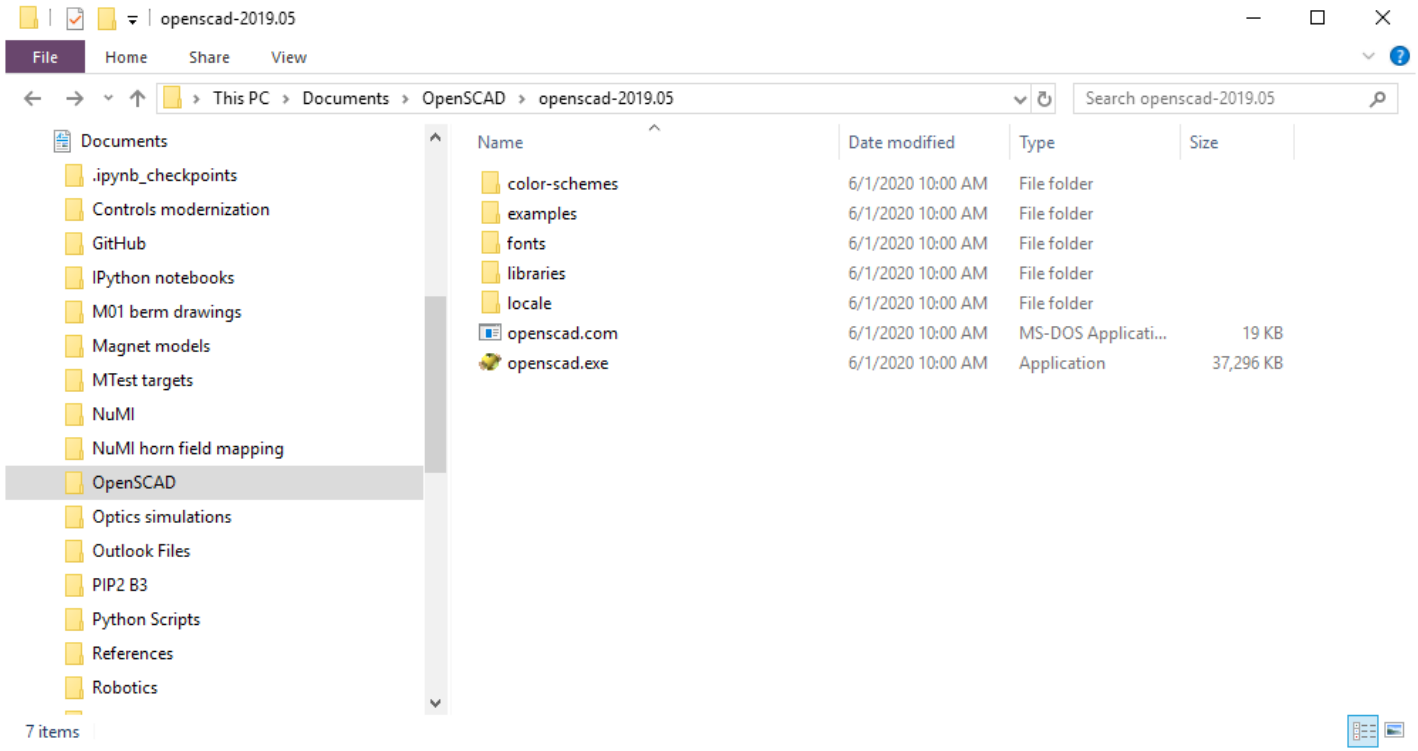
- OpenSCAD is an open-source Computer-Aided-Drafting application
- Unique because rather than focusing on a full-featured GUI for building models, it uses a C-language programming interface for the user input
- Helpful for creation of complex objects from mathematical formulae
- Input file is text, so can be created from converting other files (such as Tom's idea of the survey/floor output of a beam simulation program)
- Very fast and lightweight; doesn't even need to be installed.
- Can output to standard file formats (.STL step files) or to 3D printers!
- Great for building models that can be passed on to Engineers or Drafters, or for prototyping custom parts to be 3D printed or manufactured.

Getting started



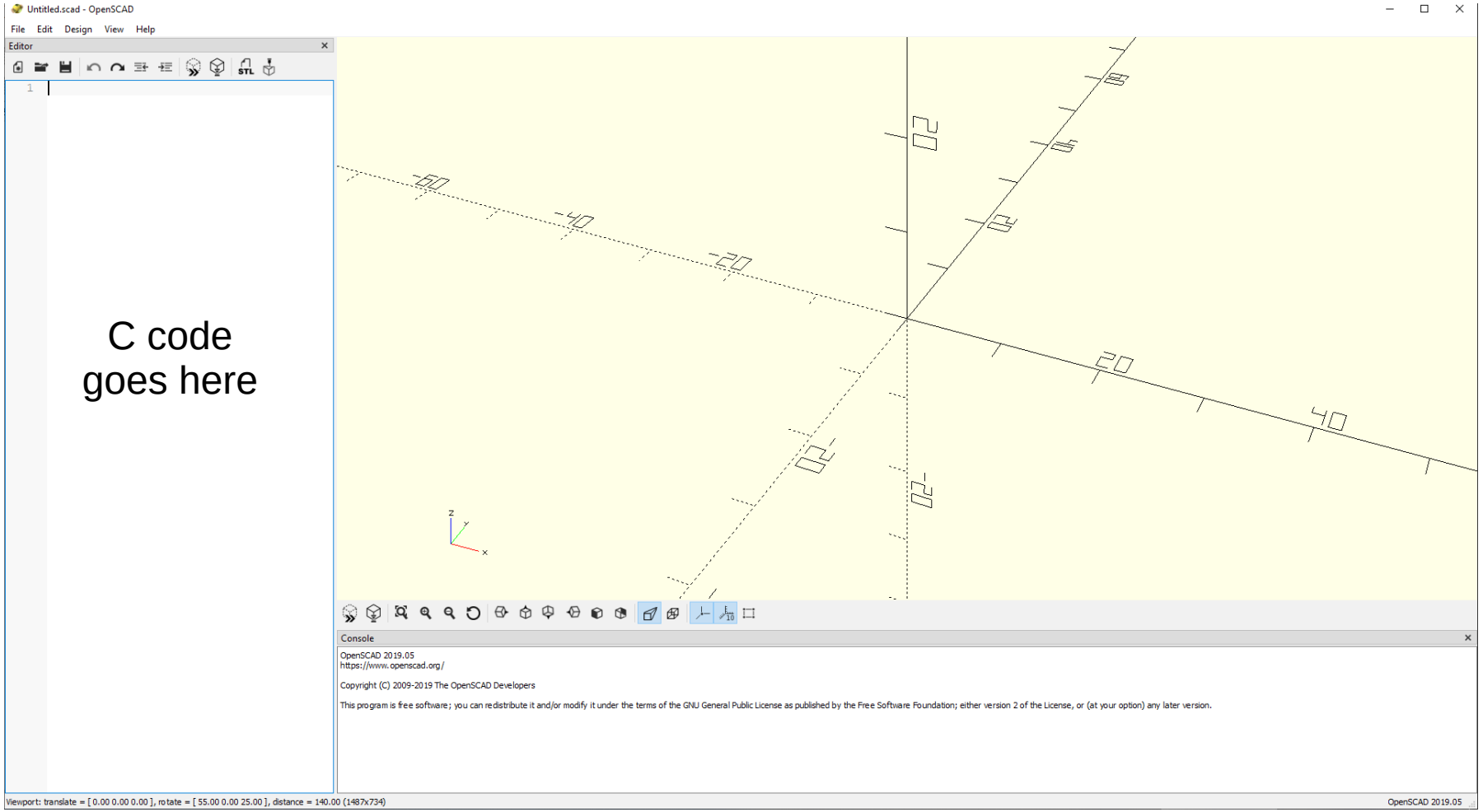
- https://www.openscad.org/downloads.html
- Widely-available installer packages for most operating systems, and can be built from source.
- Pre-compiled “portable” binaries for Windows available in “zip package” link. No need for Administrative priveliges.

Getting started (Windows)



- Download and extract the zip file.
- Start the program by running the executable “openscad.exe”
- Input files are “.scad”, but can be read by text editor.
- Lots of example .scad files under “examples” directory

New project



- A new blank project window looks like the above screenshot.
- CTRL+S saves and automatically preview-renders the model in the right panel

Cheat sheet

OpenSCAD v2019.05

Syntax

```
var = value;
var = cond ? value_if_true : value_if_false;
module name(…) { … }
name();
function name(…) = …
name();
include <….scad>
use <….scad>
```

Constants

undef	undefined value
PI	mathematical constant π (~3.14159)

Special variables

$fa	minimum angle
$fs	minimum size
$fn	number of fragments
$t	animation step
$vpr	viewport rotation angles in degrees
$vpt	viewport translation
$vpd	viewport camera distance
$children	number of module children
$preview	true in F5 preview, false for F6

Modifier Characters

*	disable
!	show only
#	highlight / debug
%	transparent / background

2D

```
circle(radius | d=diameter)
square(size,center)
square([width,height],center)
polygon([points])
polygon([points],[paths])
text(t, size, font,
     halign, valign, spacing,
     direction, language, script)
import("….ext")
projection(cut)
```

3D

```
sphere(radius | d=diameter)
cube(size, center)
cube([width,depth,height], center)
cylinder(h,r|d,center)
cylinder(h,r1|d1,r2|d2,center)
polyhedron(points, faces, convexity)
import("….ext")
linear_extrude(height,center,convexity,twist,slices)
rotate_extrude(angle,convexity)
surface(file = "….ext",center,convexity)
```

Transformations

```
translate([x,y,z])
rotate([x,y,z])
rotate(a, [x,y,z])
scale([x,y,z])
resize([x,y,z],auto)
mirror([x,y,z])
multmatrix(m)
color("colorname",alpha)
color("#hexvalue")
color([r,g,b,a])
offset(r|delta,chamfer)
hull()
minkowski()
```

Boolean operations

```
union()
difference()
intersection()
```

List Comprehensions

```
Generate [ for (i = range|list) i ]
Generate [ for (init;condition;next) i ]
Flatten [ each i ]
Conditions [ for (i = …) if (condition(i)) i ]
Conditions [ for (i = …) if (condition(i)) x else y ]
Assignments [ for (i = …) let (assignments) a ]
```

Flow Control

```
for (i = [start:end]) { … }
for (i = [start:step:end]) { … }
for (i = […,…,…]) { … }
for (i = …, j = …, …) { … }
intersection_for(i = [start:end]) { … }
intersection_for(i = [start:step:end]) { … }
intersection_for(i = […,…,…]) { … }
if (…) { … }
let (…) { … }
```

Type test functions

```
is_undef
is_bool
is_num
is_string
is_list
```

Other

```
echo(…)
render(convexity)
children([idx])
assert(condition, message)
assign (…) { … }
```

Functions

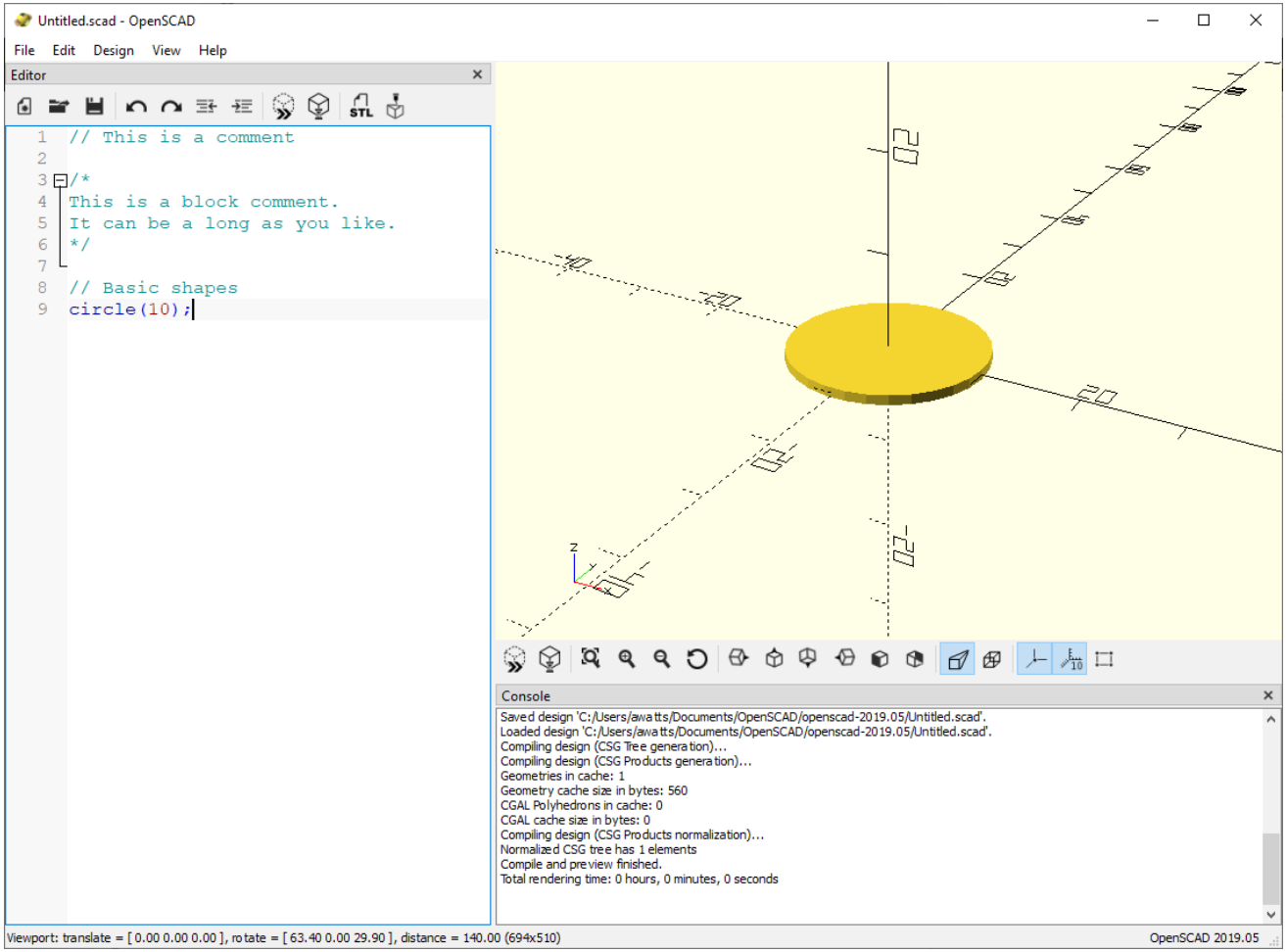
```
concat
lookup
str
chr
ord
search
version
version_num
parent_module(idx)
```

Mathematical

```
abs
sign
sin
cos
tan
acos
asin
atan
atan2
floor
round
ceil
ln
len
let
log
pow
sqrt
exp
rands
min
max
norm
cross
```

http://www.openscad.org/cheatsheet/

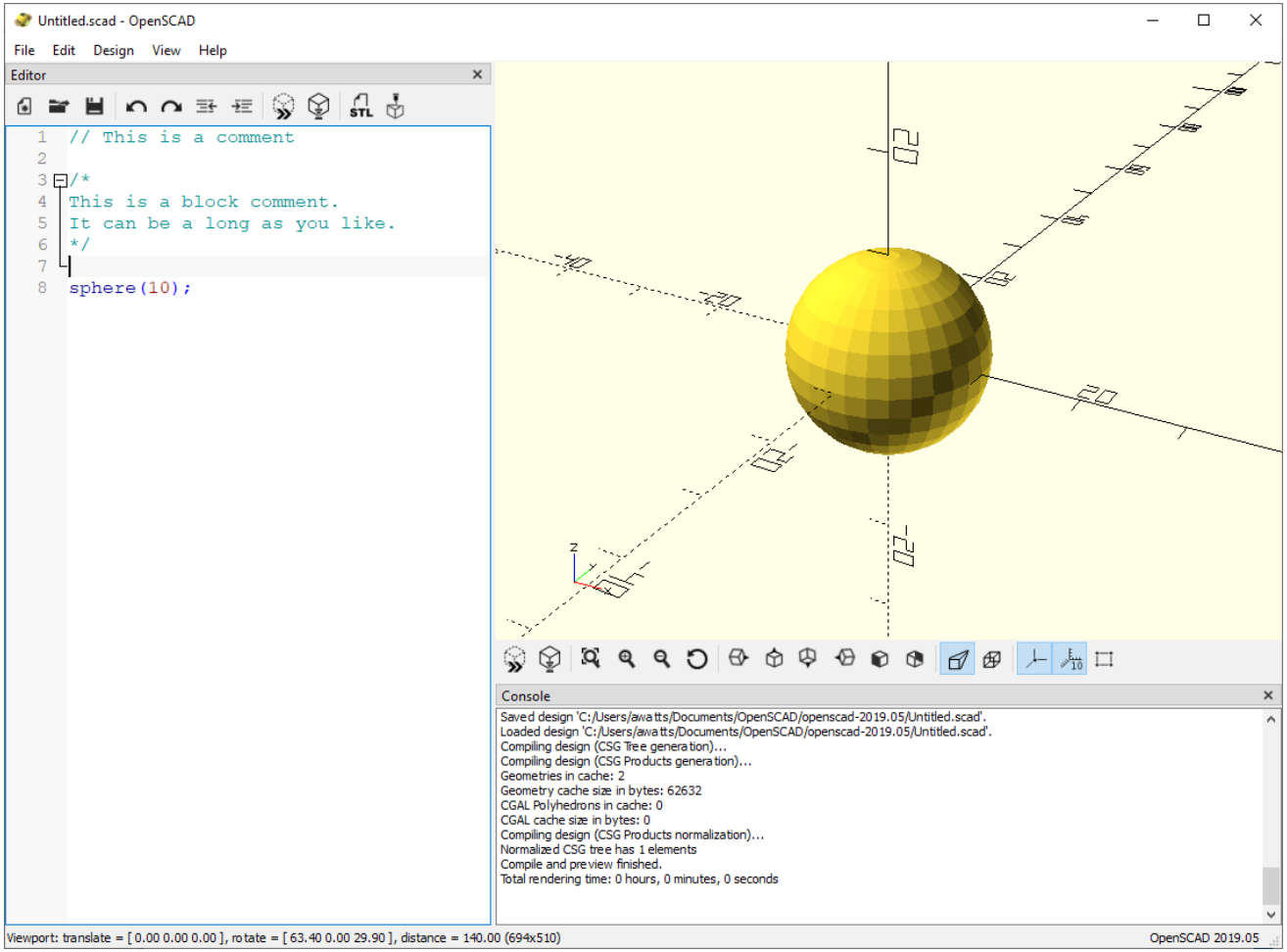
2D object



- Elements are created like calling functions in C: don't forget the semicolon!
- 2D shapes have fixed nonzero height, which I don't find useful. Instead of circle (shown above), I'd use a cylinder so I can specify the height as well.

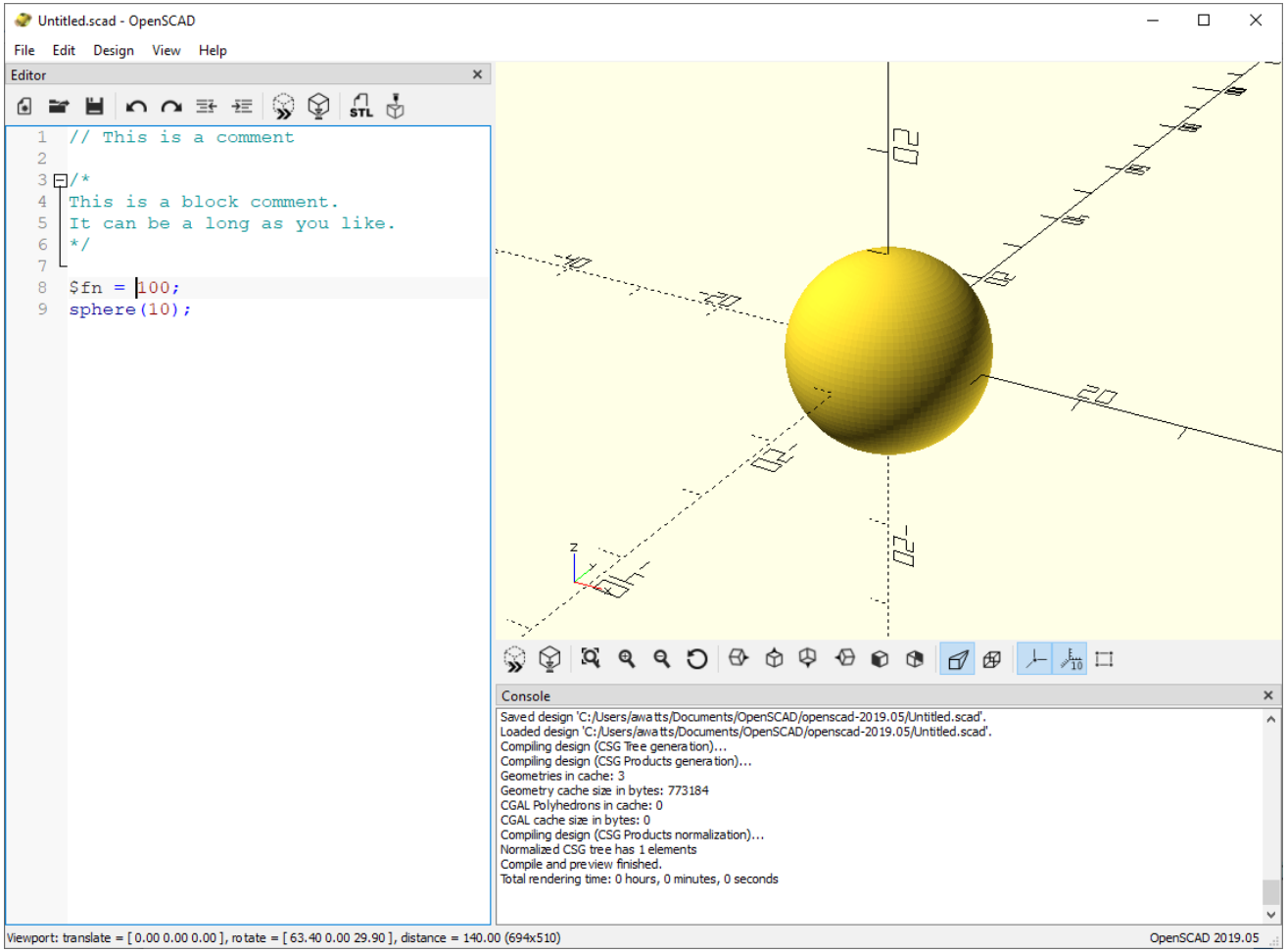
Fermilab

3D Object



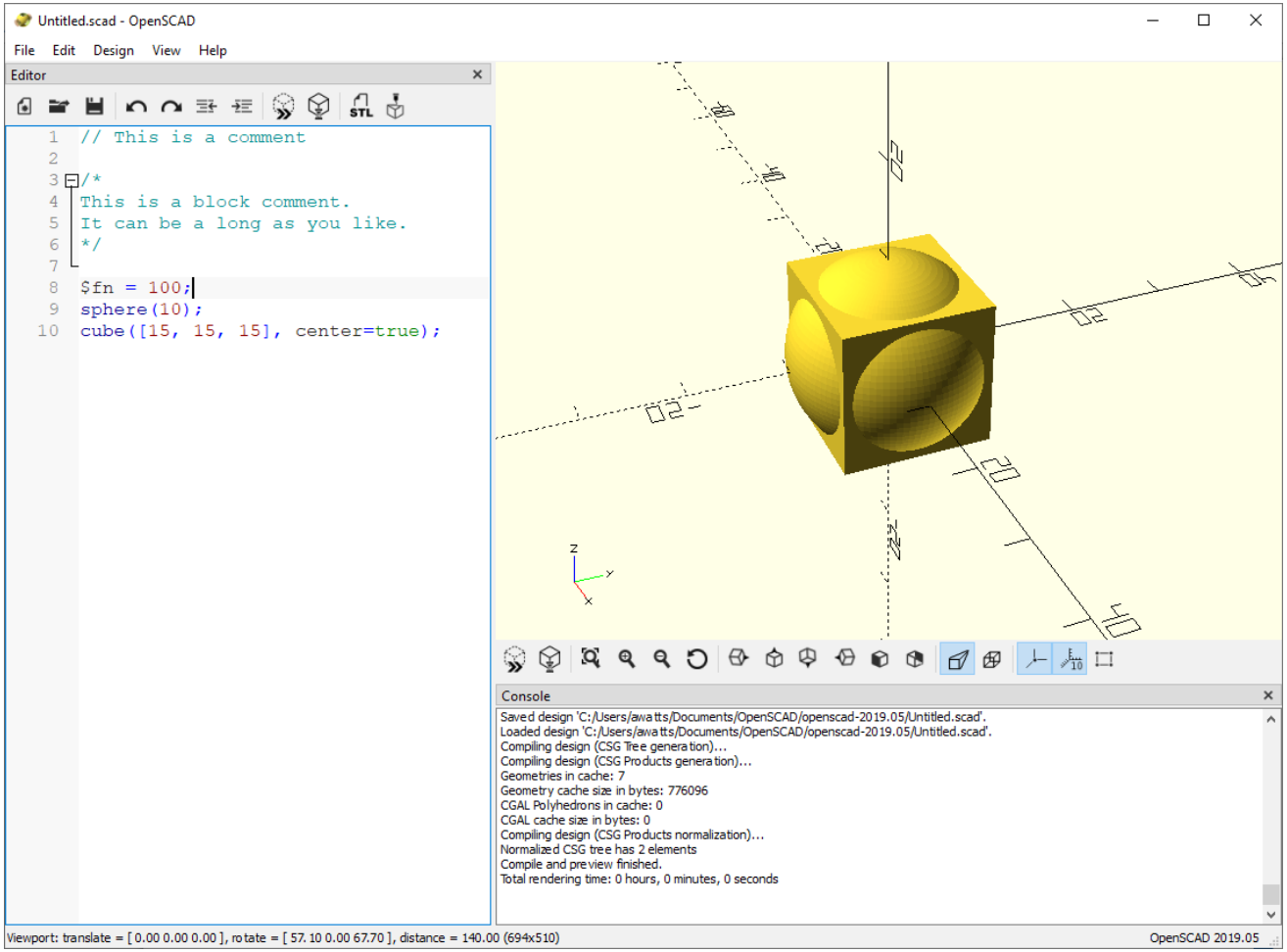
- Sphere example shown above looks a bit chunky. Can increase polygon count to smooth out if necessary.

Increasing number of polygons



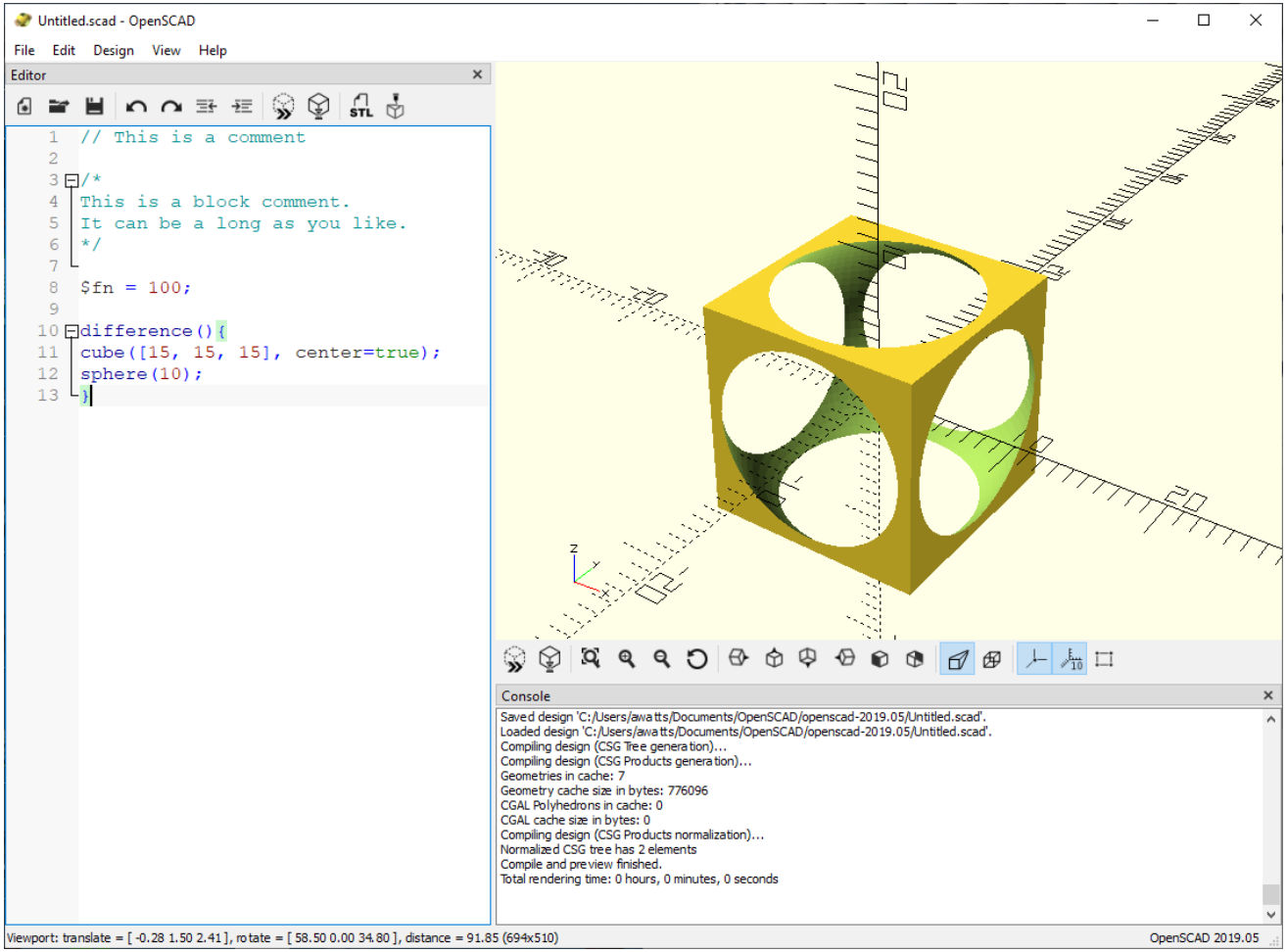
- Set system variable “$fn” to use larger number of “fragments”, smoothing out models.

Union of intersecting objects



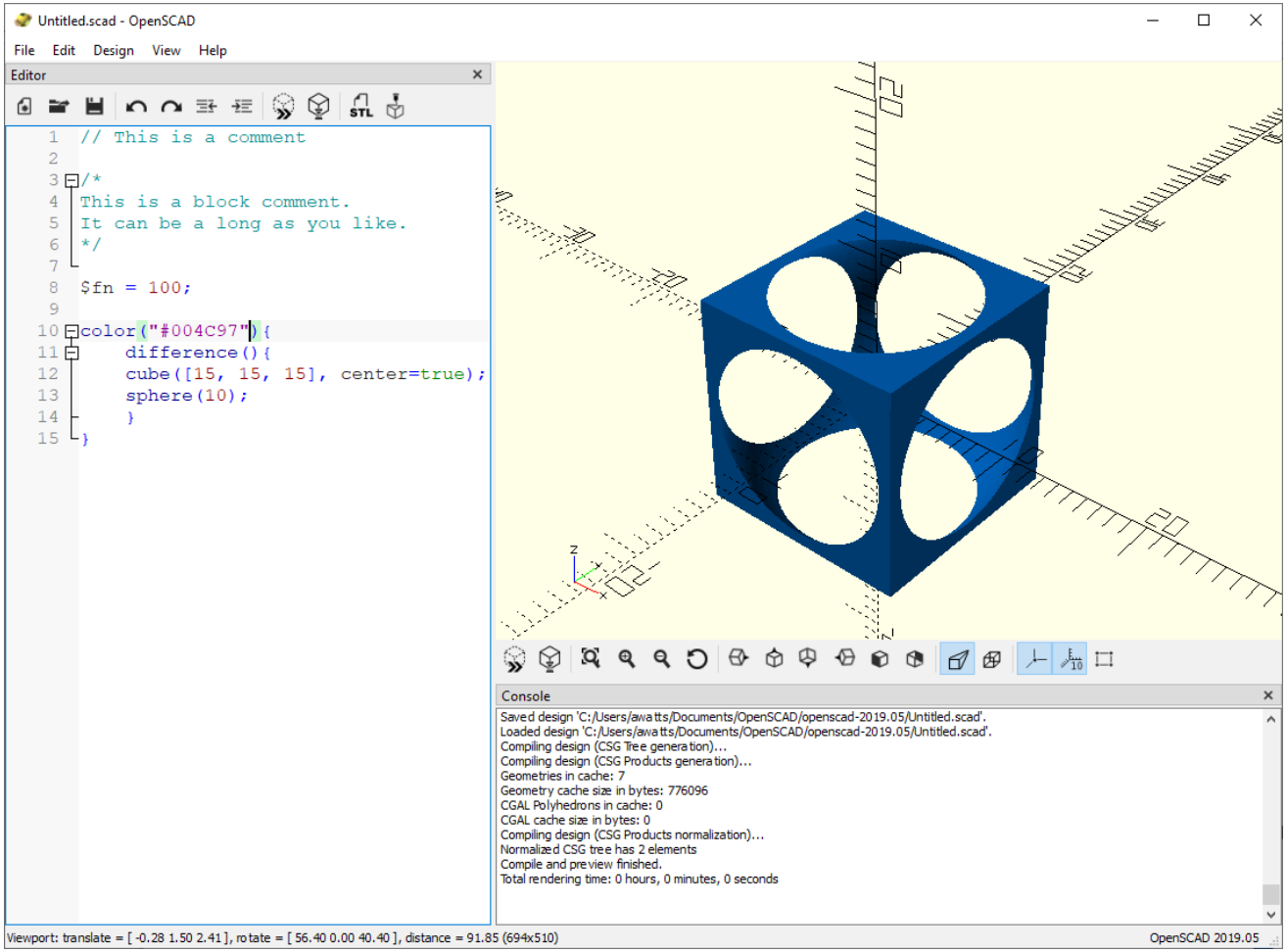
- Added a “cube”, which is really a general rectangular prism if you pass an array of x, y, and z widths. I set all widths equal to 15 units, and included the optional flag “center=true” to center the cube at the origin.
- Intersecting shapes are automatically joined together. Same result as using “union”.

Difference of intersecting objects



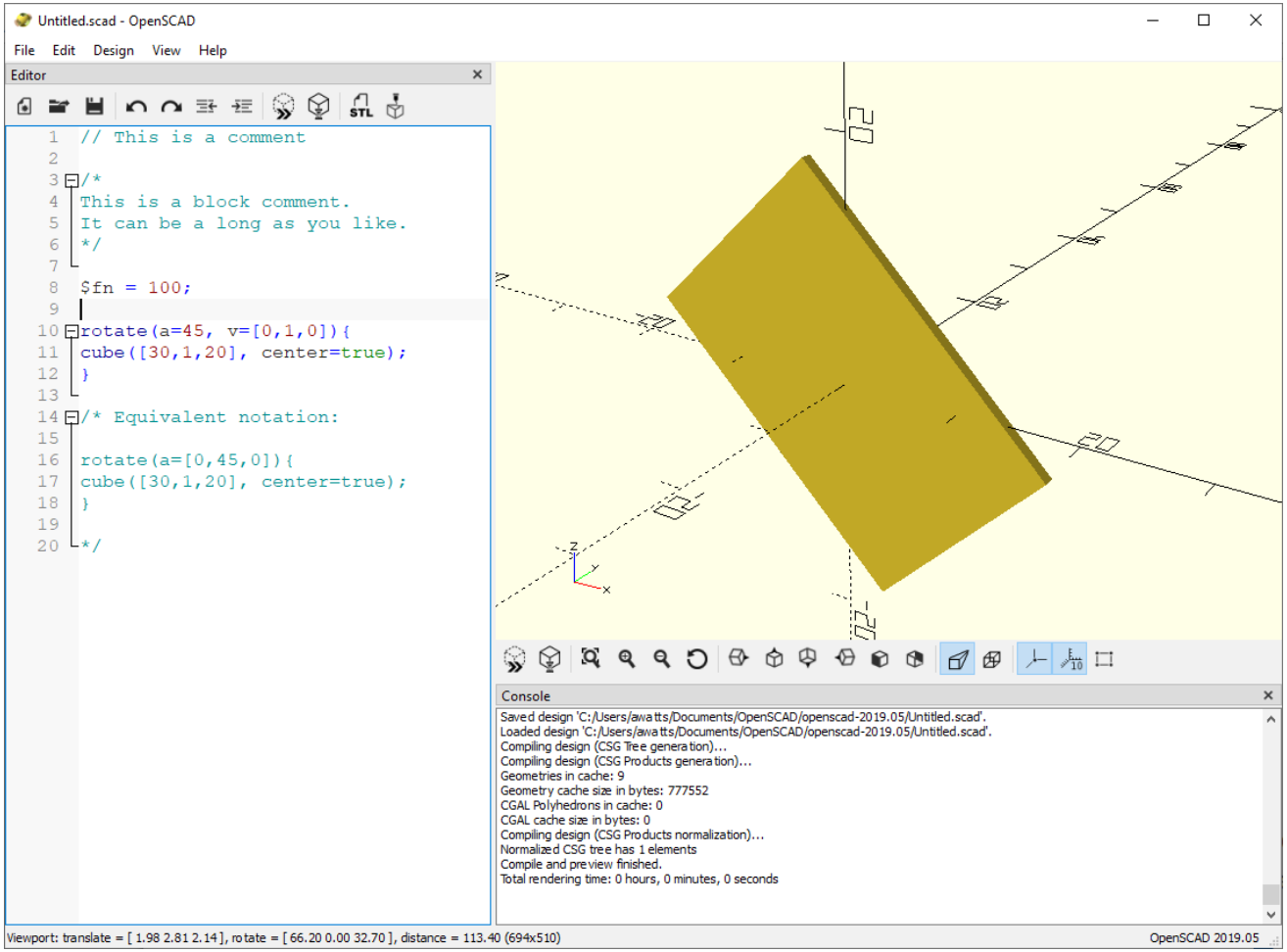
- The “difference” command subtracts the second element from the first. Useful for making holes.
- Note that for Windows version, I had to go to “View” and un-check “Thrown together” to actually show holes. Otherwise it was highlighting missing material in green.

Color



- The “color” command changes the color in the display for anything inside the curly braces.
- Accepts lots of types of arguments: shown above is the hex code for FNAL blue, for example.
- Can also set alpha value for transparency.

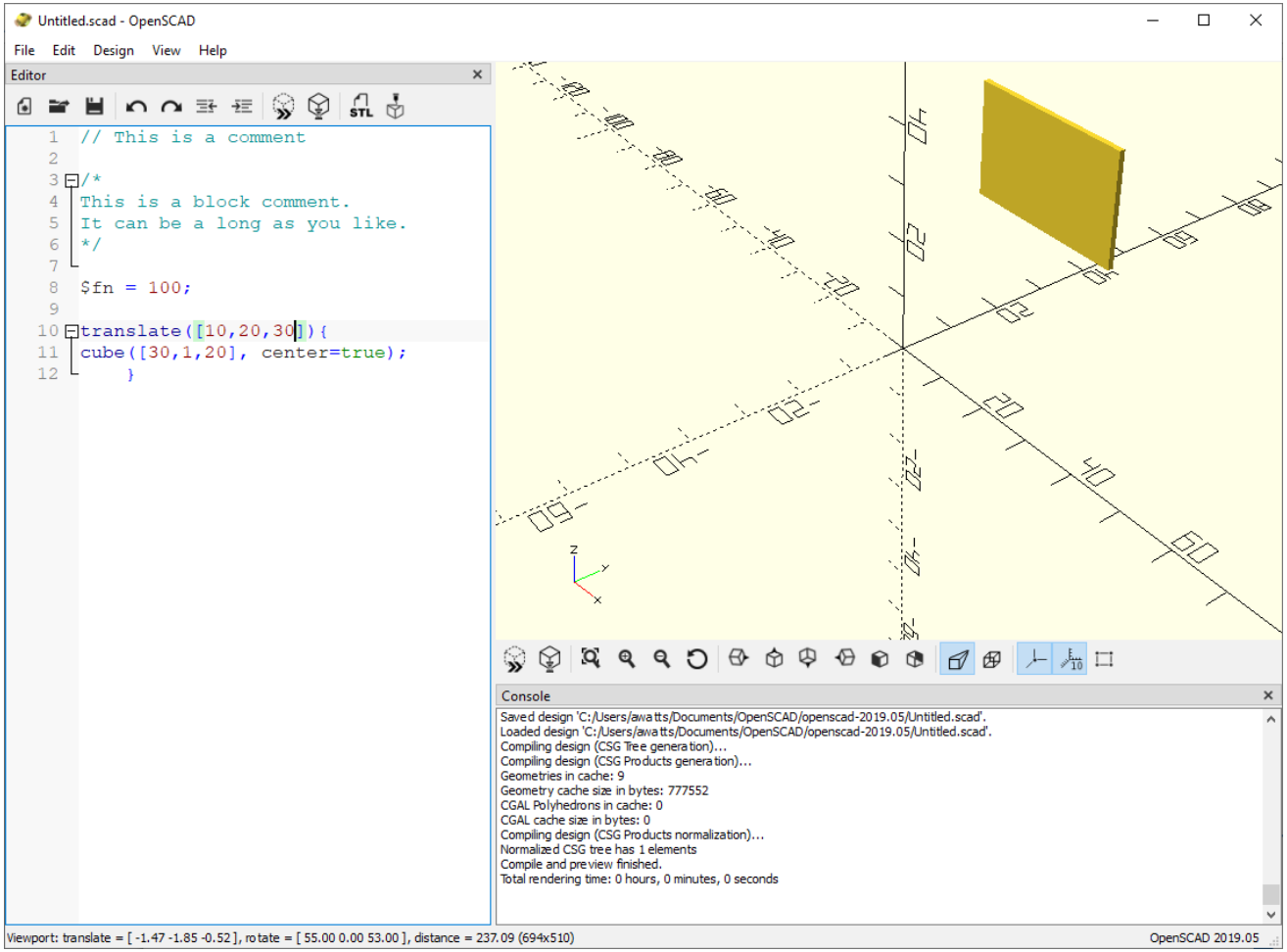
Rotate



- The “rotate” command rotates anything between its curly braces by an angle “a” in degrees about a vector “v”
- Equivalent notation can combine angle and vector together.

Fermilab

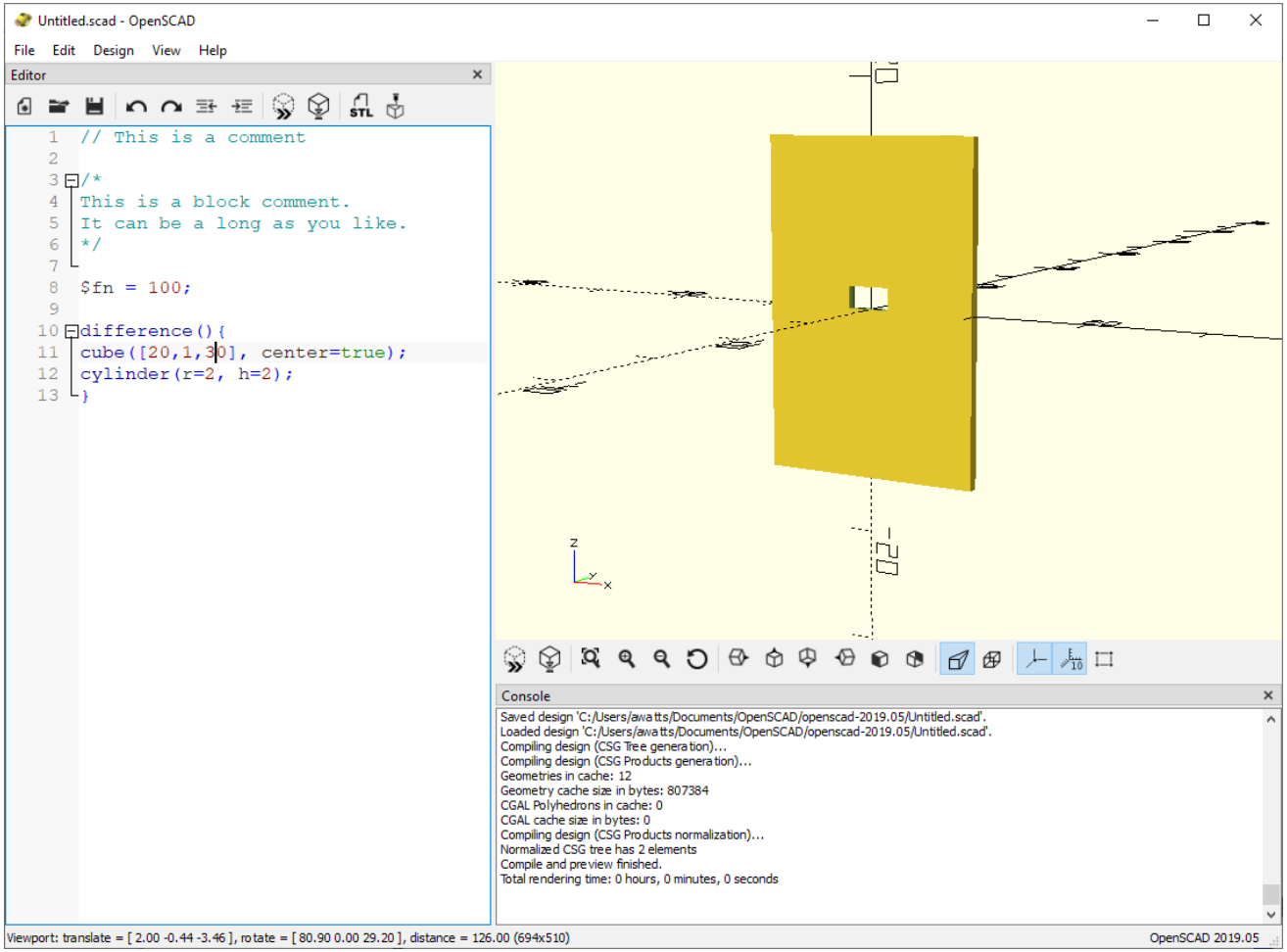
Rotate



- The “translate” command translates anything between the curly braces along each axis (x,y,z) as specified by the array argument.
- In the above case, I’ve translated the “cube” by 10 units along the x-axis, 20 units along y, and 30 units along z.

Fermilab

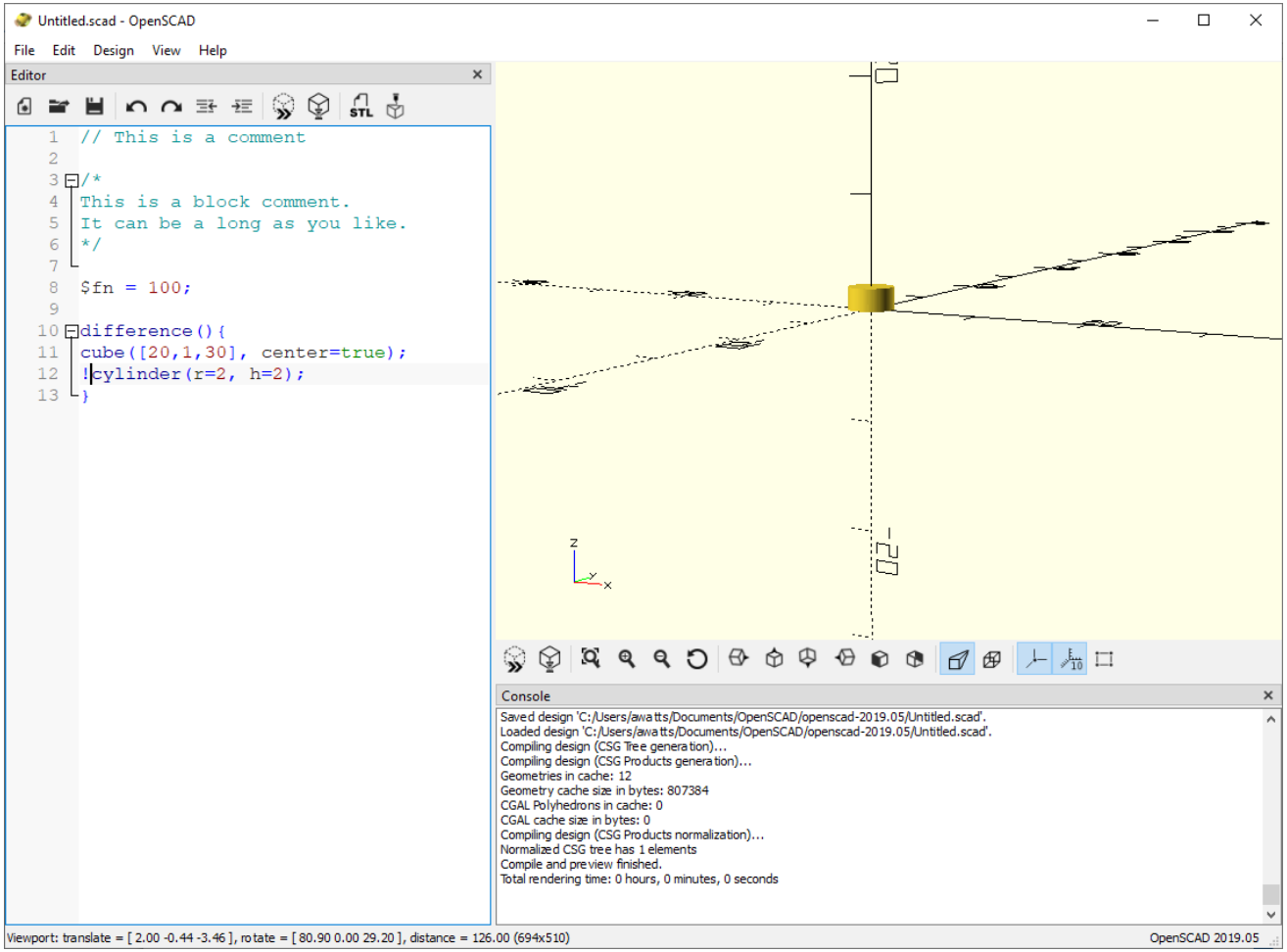
Troubleshooting



- I tried to punch a hole in this plate, but it didn't turn out the way I want.
- Rather than deleting code to see what happened, I can use "!" to plot only that line
- Useful for complicated shapes with lots of lines

Fermilab

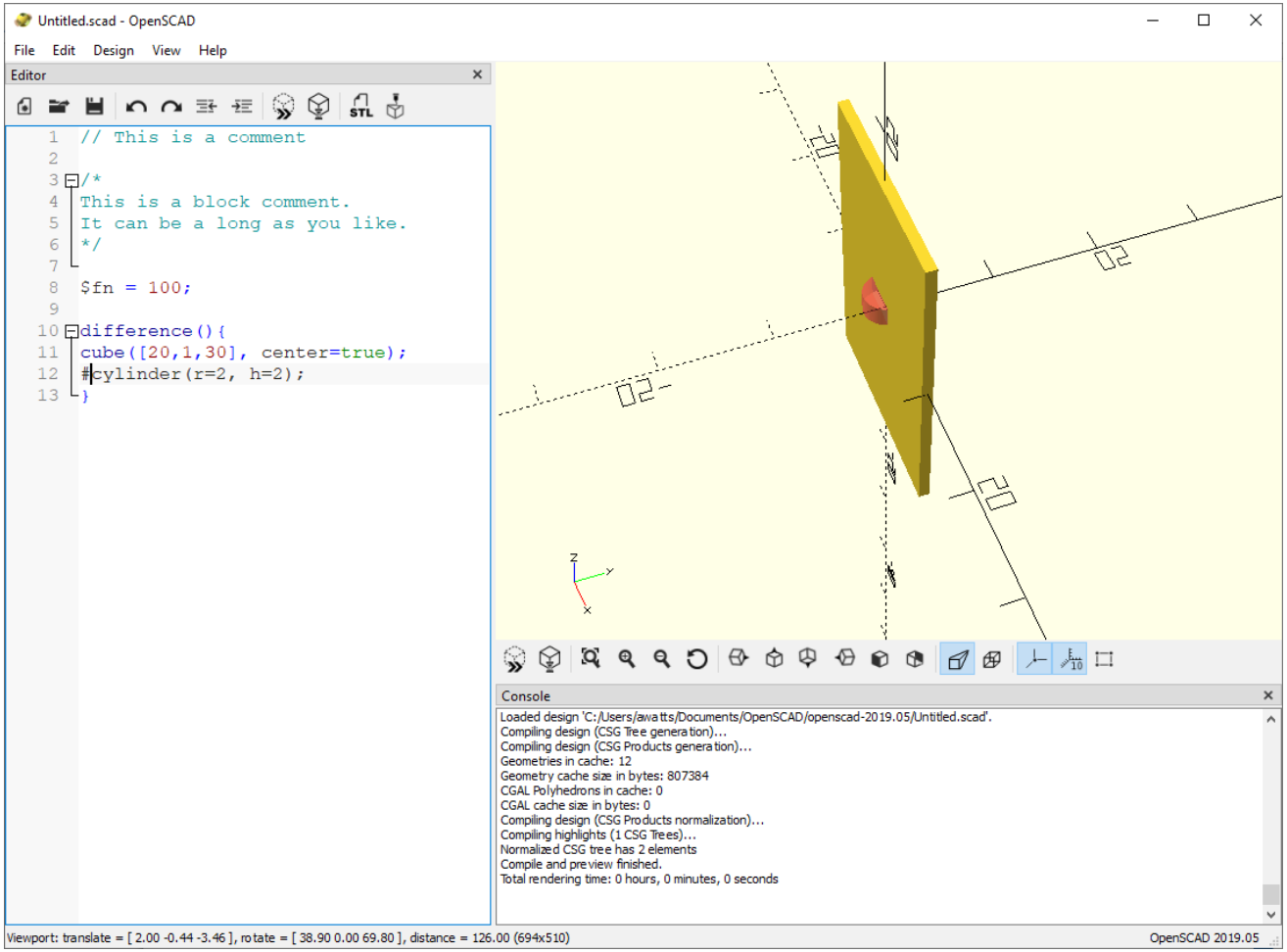
Troubleshooting



- I tried to punch a hole in this plate, but it didn't turn out the way I want.
- Rather than deleting code to see what happened, I can use "!" to plot only that line
- Useful for complicated shapes with lots of lines

Fermilab

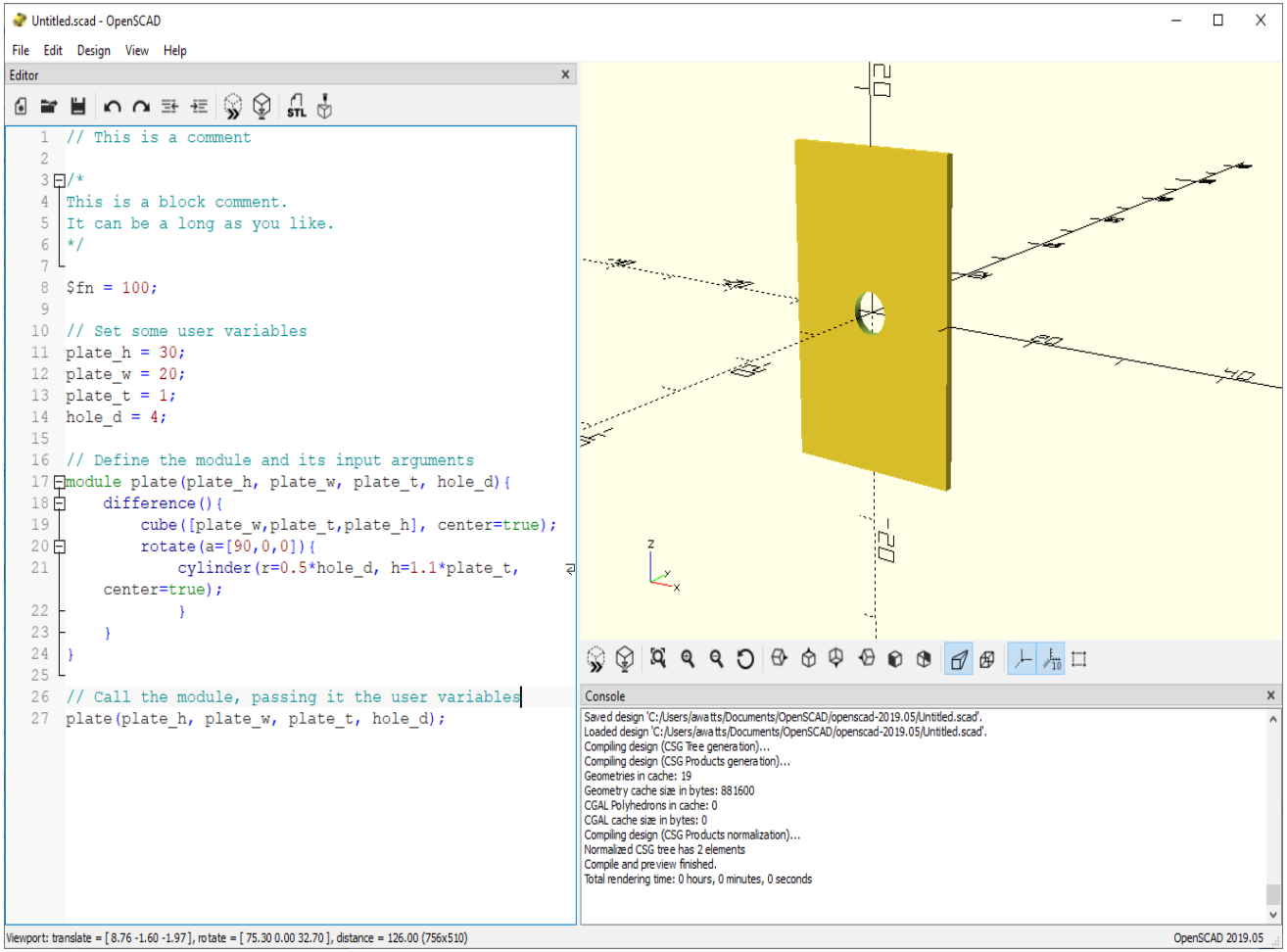
Troubleshooting



- I tried to punch a hole in this plate, but it didn't turn out the way I want.
- I can also use a "#" to highlight that line.
- Looks like I forgot to rotate the cylinder to punch the hole in the plate

Fermilab

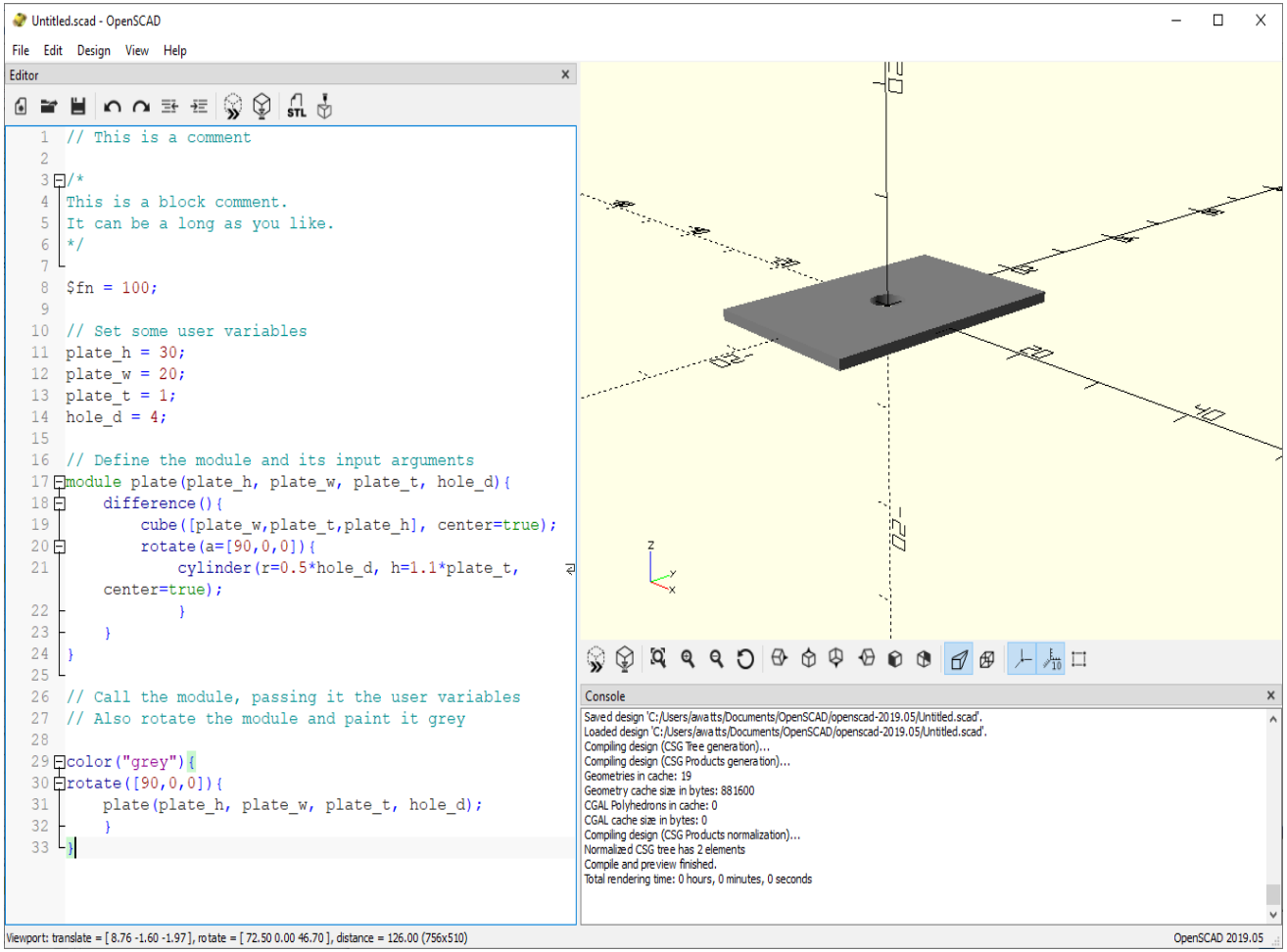
Modules, arguments, and variables



- Create a module that is defined as some combination of shapes, and that module can accept variables as arguments.
- Once module is defined, it can be called while passing either numbered arguments, or user variables.

Fermilab

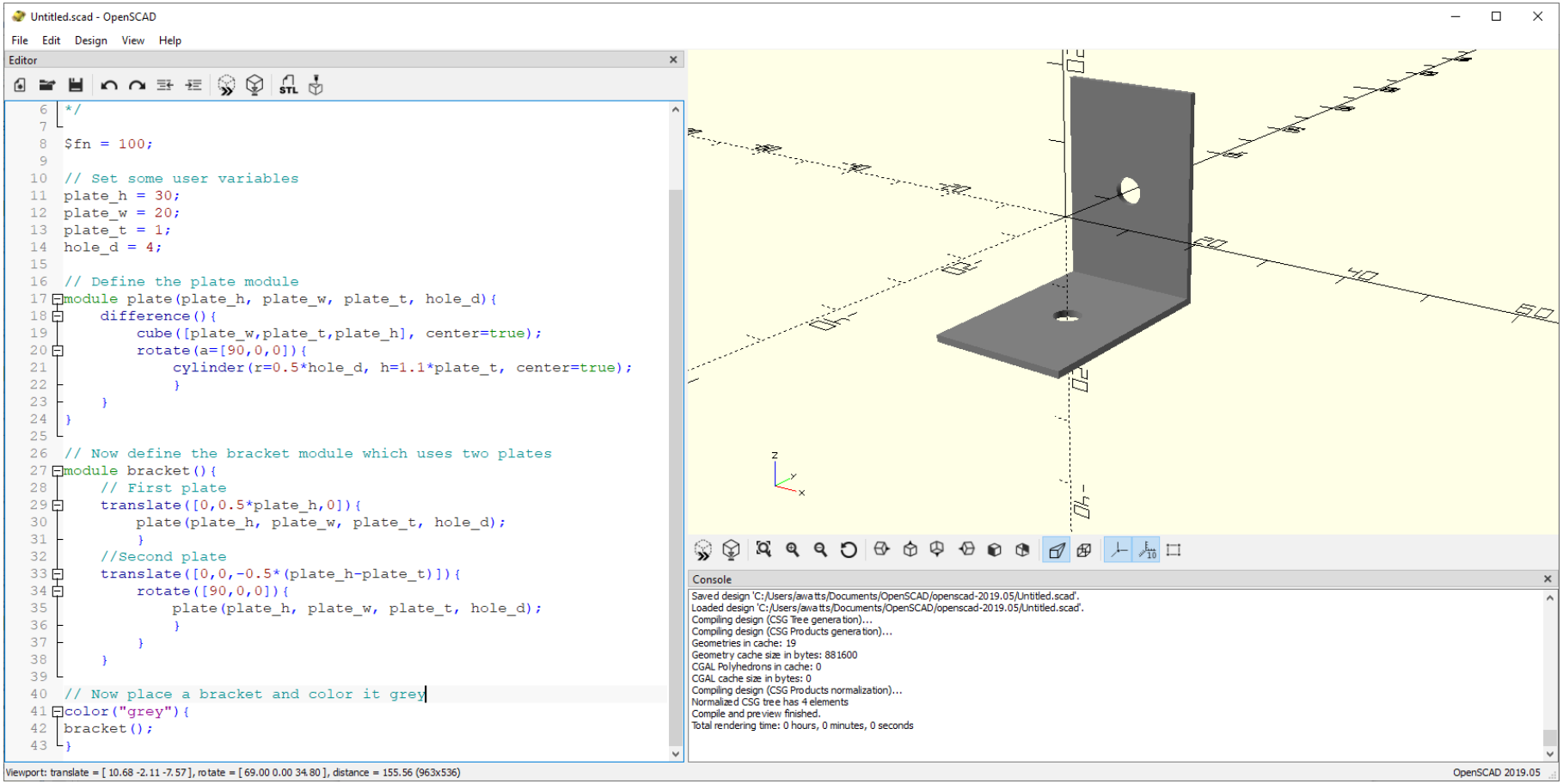
Modules, arguments, and variables



- The same commands we've used before (rotate, translate, color) can be used on modules.

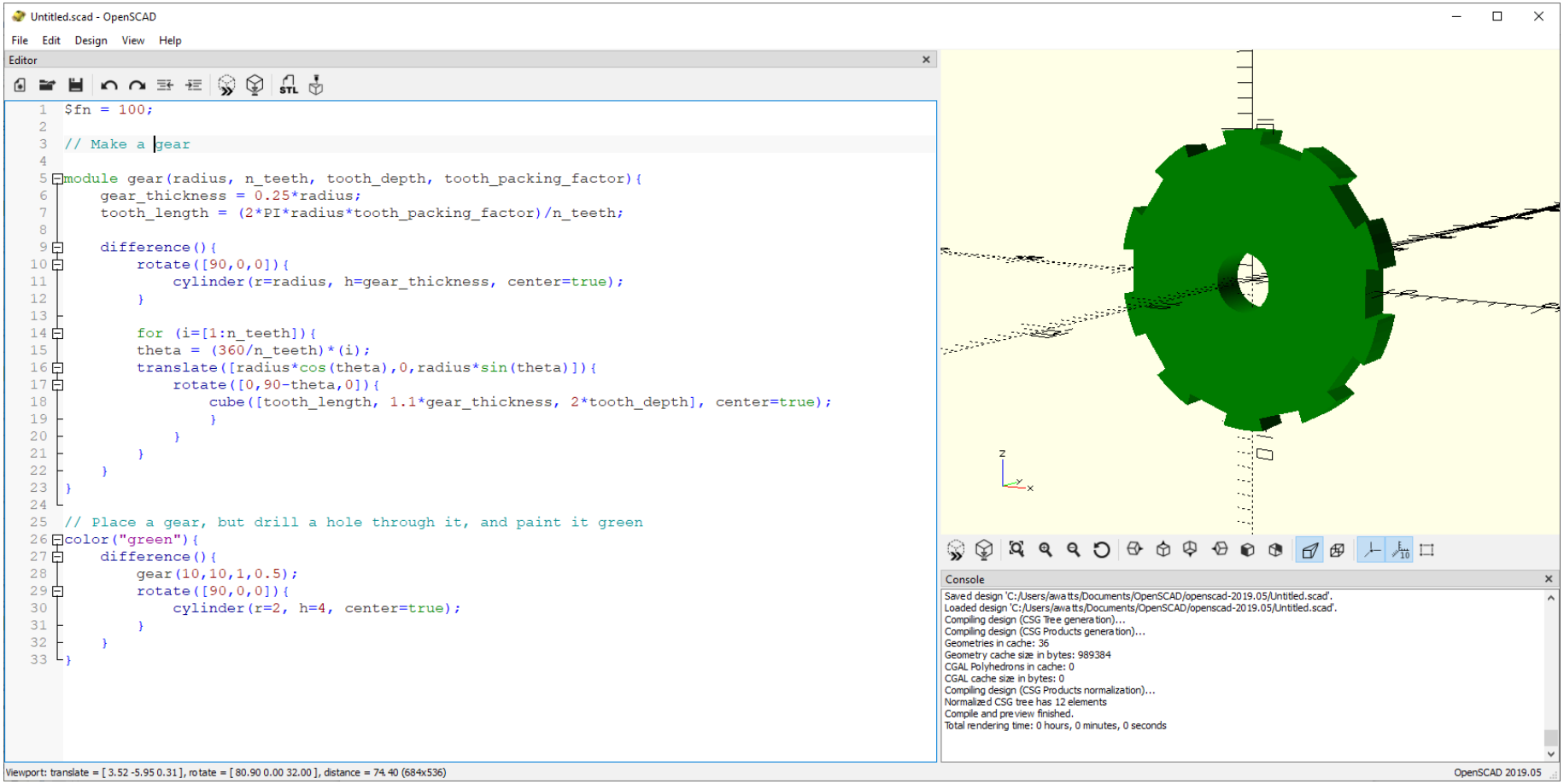
Fermilab

A module of modules: the bracket



- We can create a new module out of a combination of other modules.
- The module “bracket” is made up of two “plate” modules
- Now when we place the bracket by calling its module name, we can apply the same previous commands to it, like translate, rotate, and color.

Putting it all together, with a loop: the gear



- We can write “for” loops to automate placing several copies of the same shape, like the teeth cutouts around the gear’s circumference.
- Once a module is created, we can still use the “difference” command on it.